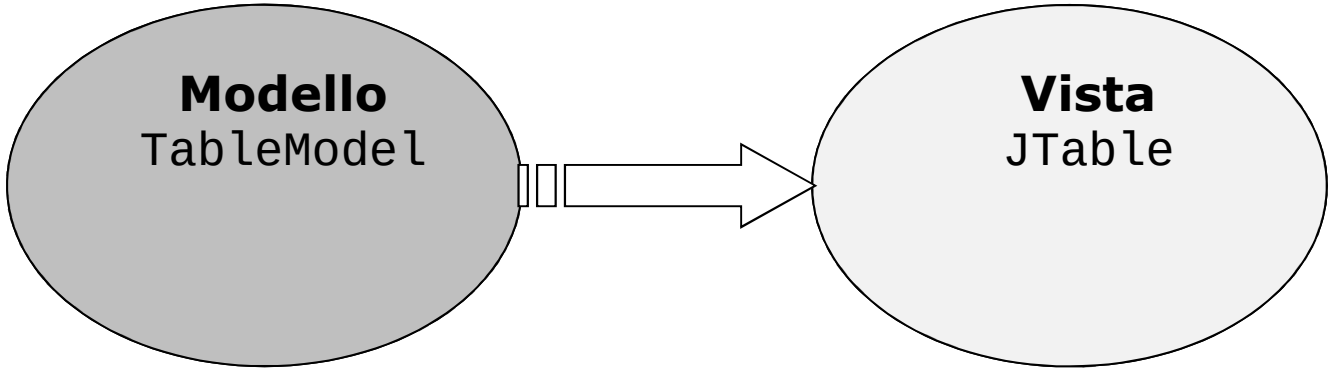
JTable

- In alcune applicazioni è necessario mostrare le informazioni in formato tabellare

Nome	Cognome	Indirizzo	Telefono
Mario	Bianchi	Via Roma, 12	059/1111111
Franco	Rossi	Via Milano, 33	059/2222222

JTable - JTable e TableModel

- Swing mette a disposizione una classe che implementa il componente tabella: **`JTable`**
- Ogni tabella implementata con **`JTable`** recupera i dati da rappresentare tramite un **modello**, istanza di una classe che implementa l'interfaccia **`TableModel`**

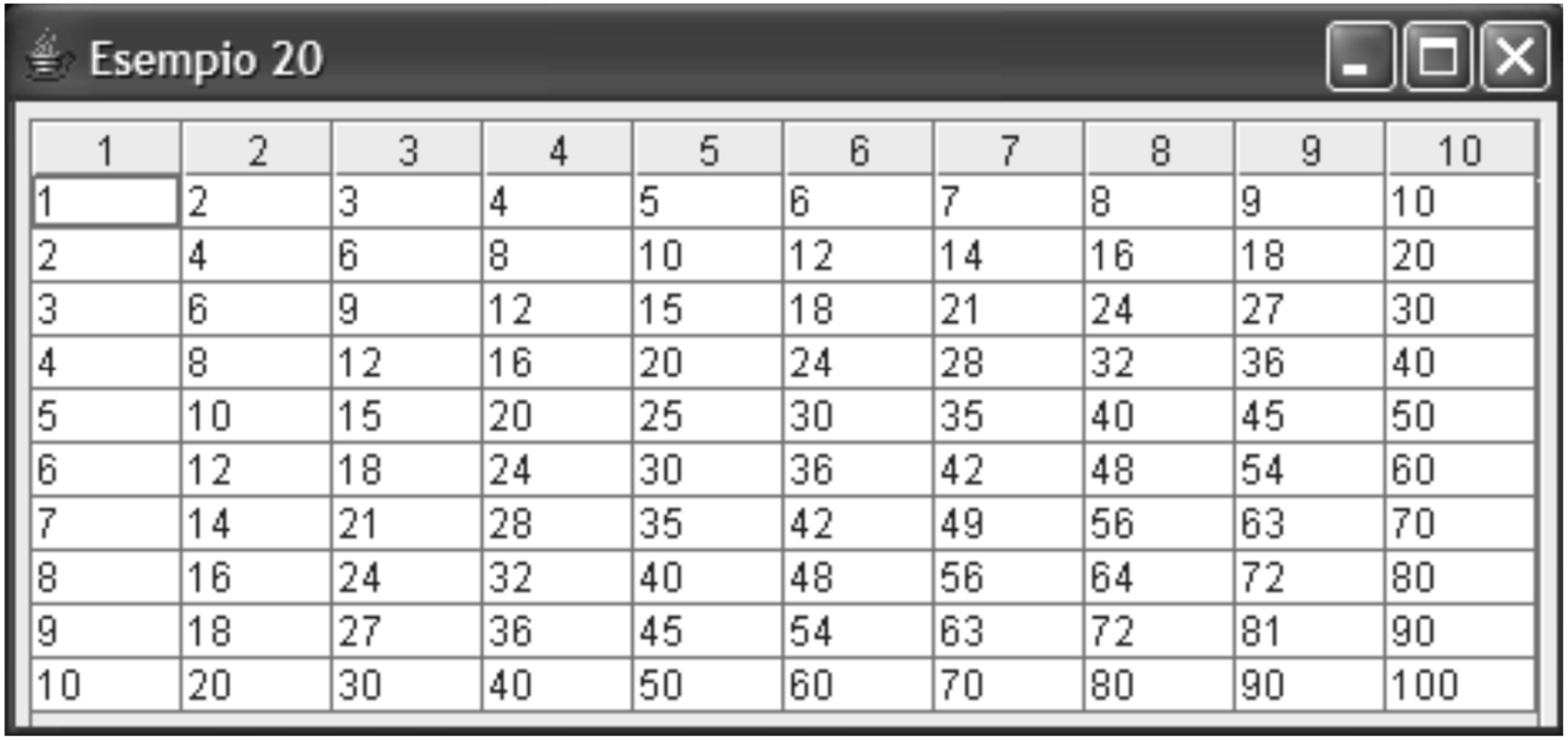


JTable - TableModel

- Questa interfaccia mette a disposizione **metodi** per sapere quante righe/colonne servono, qual è il valore di ogni singola cella, se le celle sono editabili, ecc.
- È disponibile anche una classe astratta, **`AbstractTableModel`**, che implementa la maggior parte dei metodi di **`TableModel`**, e lascia al programmatore da implementare solo i tre metodi:
  - **public int getRowCount();**
  - **public int getColumnCount();**
  - **public Object getValueAt(int row, int column);**

JTable – Esempio: le tabelline

- Vogliamo visualizzare le tebelline delle moltiplicazioni



1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

JTable – Esempio: modello e vista

- Modello dei dati
  - Il modello è dato da una classe che implementa **`TableModel`** (o meglio, estende **`AbstractTableModel`**)
  - Il contenuto di ogni cella è il **prodotto** dell'indice di riga per l'indice di colonna
  - Siccome gli indici partono da 0, dovremo incrementarli entrambi
  - Il modello definisce anche l'**intestazione** delle colonne
- Vista dei dati
  - La tabella che mostra i dati è una istanza di **`JTable`**, che viene inizializzata con una istanza della classe precedente

JTable – Esempio: implementazione del modello

```
import javax.swing.table.AbstractTableModel;

public class MyTableModel extends AbstractTableModel {

	// ritorna il numero di colonne
	public int getColumnCount() { return 10; }

	// ritorna il numero di righe
	public int getRowCount() { return 10;}

	// ritorna il contenuto di una cella
	public Object getValueAt(int row, int col)
	{
		// ritorna il prodotto (come oggetto)
		return new Integer((row+1)*(col+1));
	}
```

JTable – Esempio: implementazione del modello (2)

```
	// ritorna il nome della colonna
	public String getColumnName(int col) {
		// e' il numero di colonna
		return Integer.toString(col+1);
	}

	// specifica se le celle sono editabili
	public boolean isCellEditable(int row, int
col) {
		// nessuna cella editabile
		return false;
	}
}
```

JTable – Esempio: implementazione della vista

```
import javax.swing.*;

public class Es20Panel extends JPanel {
  public Es20Panel() {
            // crea il modello di dati
            TableModel dataModel = new MyTableModel();
            // crea la tabella
            JTable t = new JTable(dataModel);
            // aggiunge la tabella ad uno ScollPane
            JScrollPane scrollpane = new JScrollPane(t);
            // aggiunge lo ScrollPane al pannello
            add(scrollpane);
  }
}
```

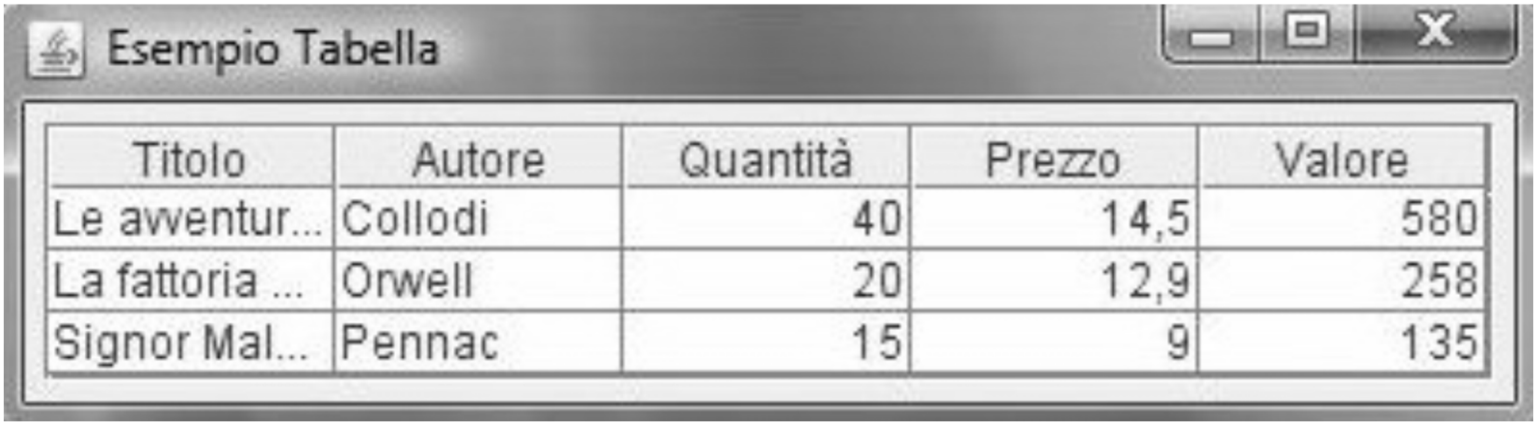
- Il pannello viene poi inserito in un **`JFrame`** come al solito

JTable – Eventi di tabella

- Naturalmente, ogni azione dell'utente sulla tabella genera un evento che può essere ascoltato da un ascoltatore
- La classe **`JTable`** stessa implementa diversi ascoltatori
  - CellEditorListener
  - TableModelListener
  - ListSelectionListener
  - TableColumnModelListener
- Non è sempre necessario implementare i metodi degli ascoltatori
  - La tabella rimane una vista dei dati contenuti nel modello

JTable – Esempio 2

- Vogliamo visualizzare una lista di libri, ognuno con titolo, autore, quantità, prezzo e valore totale, dato dal prezzo moltiplicato per la quantità
- Diamo la possibilità all'utente di modificare quantità e prezzo



Titolo	Autore	Quantità	Prezzo	Valore
Le avventur...	Collodi	40	14,5	580
La fattoria ...	Orwell	20	12,9	258
Signor Mal...	Pennac	15	9	135

JTable – Esempio 2: la classe Book

- Ogni libro è rappresentato da una istanza della classe Book

```
public class Book {
	public String title; // titolo
	public String author; // autore
	public int quantity; // quantita'
	public float price; // prezzo

	// costruttore
	public Book(String title, String author, int
quantity, float price) {
		this.title = title;
		this.author = author;
		this.quantity = quantity;
		this.price = price;
	}
}
```

JTable – Esempio 2: il modello dei dati

```
import java.util.Vector;
import javax.swing.table.AbstractTableModel;

public class VectorTableModel extends AbstractTableModel {
        Vector v = null;
        // intestazioni delle colonne
        String[] ColName = {"Titolo", "Autore", "Quantità",
"Prezzo", "Valore" };

        public VectorTableModel(Vector v) {
                this.v = v; // inizializzato con il vettore
        }
        /** il numero di colonne */
         public int getColumnCount()
        { return ColName.length; }
```

JTable – Esempio 2: il modello dei dati (2)

```
/** numero righe = dimensione del vettore */
 public int getRowCount() { return v.size(); }

 /** ritorna il contenuto di una cella */
 public Object getValueAt(int row, int col) {
      // seleziona il libro
      Book b = (Book)v.elementAt(row);
      // la stringa corrispondente alla colonna
      switch (col){
             case 0: return b.title;
             case 1: return b.author;
             case 2: return b.quantity;
             case 3: return b.price;
             case 4: return b.price * b.quantity;
             default: return "";
      }
 }
```

JTable – Esempio 2: il modello dei dati (3)

```
 /**  ritorna il nome della colonna */
  public String getColumnName(int col) {
            return ColName[col];
      }

  /** ritorna il tipo dei valori
    * serve per allineare correttamente i
numeri */
  public Class getColumnClass(int col) {
            return getValueAt(0,
col).getClass();
  }
```

JTable – Esempio 2: il modello dei dati (4)

- Alcune colonne sono editabili

```
/** specifica se le celle sono editabili */
      public boolean isCellEditable(int row, int
col)  {
            if ((col == 2) || (col == 3))
                  // solo la quantita' e il prezzo
                  // sono modificabili
                  return true;
            else
                  // nessuna altra cella editabile
                  return false;
      }
```

JTable – Esempio 2: modificare i dati

- Per modificare i valori del modello è necessario implementare il metodo **setValueAt**()
  - viene invocato dalla tabella ogni volta che l'utente modifica un valore in una cella
- Conoscendo la **riga** della cella modificata, possiamo recuperare il libro interessato
- Conoscendo la **colonna**, possiamo sapere quale attributo è stato modificato
- Dopodiché informiamo la tabella delle modifiche inviando un evento tramite il metodo **fireTableDataChanged**()
- Si noti che va aggiornata la vista non solo della cella modificata dall'utente, ma anche del campo "Valore" che viene calcolato in base a quantità e prezzo

JTable – Esempio 2: modificare i dati (2)

```
/** metodo per gestire le modifiche dell'utente */
       public void setValueAt(Object value, int row, int
col) {
              Book b = (Book)v.elementAt(row);
         if (col == 2)
              // modifica la quantita'
              b.quantity = ((Integer)value).intValue();
         if (col == 3)
              // modifica il prezzo
              b.price = ((Float)value).floatValue();
         // notifica il cambiamento
         fireTableDataChanged();
     }
}
```

JTable – Esempio 2: il pannello

```
import java.util.Vector;
 import javax.swing.*;
import javax.swing.table.TableModel;

public class PannelloTabellaVettore extends JPanel {
        public PannelloTabellaVettore () {
                // predispone il vettore
                Vector v = new Vector(3);
                Book b1 = new Book("Le avventure di Pinocchio",
"Collodi", 40, 14.50F);
                Book b2 = new Book("La fattoria degli animali",
"Orwell", 20, 12.90F);
                Book b3 = new Book("Signor Malaussene", "Pennac",
15, 9.00F);
                v.add(b1);
                v.add(b2);
                v.add(b3);
```

JTable – Esempio 2: la tabella

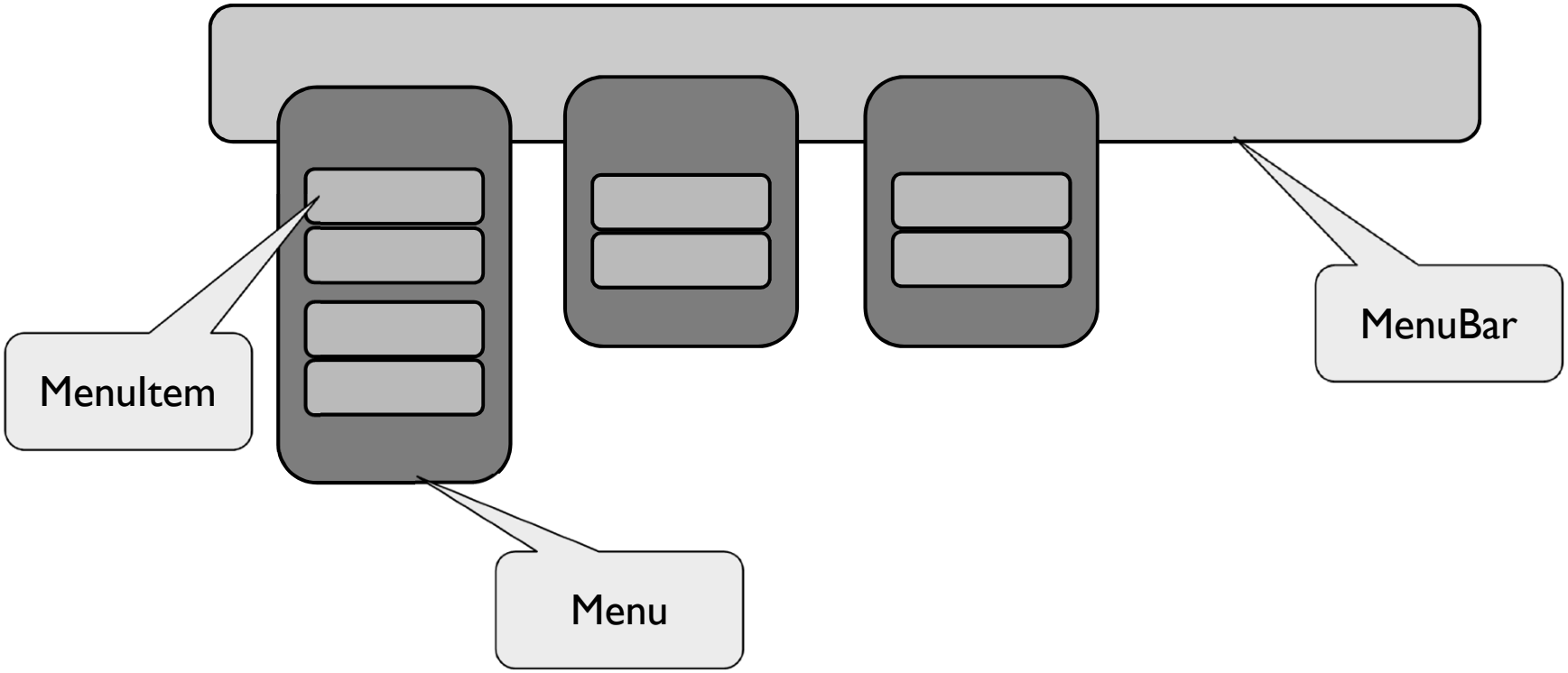
```
            // crea il modello di dati
            // a partire dal vettore
            TableModel dataModel = new
VectorTableModel(v);
            // crea la tabella
            JTable t = new JTable(dataModel);
            // imposta la dimensione della
visualizzazione
            t.setPreferredScrollableViewportSize(

                                t.getPreferredSize());
            // aggiunge la tabella ad uno ScollPane
        JScrollPane scrollpane = new JScrollPane(t);
    // aggiunge lo ScrollPane al pannello
            add(scrollpane);
      }
}
```

Menu

- Swing mette a disposizione alcune classi per creare i menu della applicazione
- **`MenuItem`**: rappresenta la voce del menu
- **`Menu`**: rappresenta un menu
- **`MenuBar`**: rappresenta la barra dei menu

Menu - struttura



Menu – una applicazione di esempio

- Implementiamo una applicazione che ha un menu per incrementare e decrementare un **contatore**
- Nota: per semplicità la label è aggiunta direttamente al frame, ma sarebbe meglio introdurre un pannello

Menu - esempio

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class MenuFrame extends JFrame
implements ActionListener {

  private JLabel l;
  private Contatore c;

  public MenuFrame() {
    this("");
  }
```

Menu – esempio (2)

```
public MenuFrame(String arg0) {
super(arg0);

c = new Contatore();

MenuItem m11 = new MenuItem("Incrementa");
MenuItem m12 = new MenuItem("Decrementa");
Menu m1 = new Menu("Contatore");
m1.add(m11);
m1.add(m12);
MenuBar mb = new MenuBar();
mb.add(m1);
this.setMenuBar(mb);
```

Menu – esempio (3)

```
  m11.addActionListener(this);
  m12.addActionListener(this);

  l = new JLabel("Valore: "+c.getVal());
  this.add(l);


this.setDefaultCloseOperation(EXIT_ON_CLOSE)
;

}
```

Menu – esempio (4)

```
public void actionPerformed(ActionEvent e) {
 if (e.getActionCommand().equals("Incrementa"))
  c.inc();
 if (e.getActionCommand().equals("Decrementa"))
  c.dec();

 l.setText("Valore: "+c.getVal());
}
}
```

Layout

- La disposizione dei componenti all'interno di un pannello può essere fatta:
  - Affidando la disposizione ad un **layout manager**
  - Specificando la posizione assoluta (**setBounds()** o **setLocation()**)
- Un layout manager dispone i componenti inseriti nel pannello secondo criteri predefiniti e/o specificati dal programmatore

Layout Manager

- Per impostare un layout manager si usa il metodo

  setLayout(LayoutManager mgr)

- Dove `mgr` può essere un oggetto delle seguenti classi:
  - **`FlowLayout`**, che dispone i componenti in sequenza (default)
  - **`BorderLayout`**, che dispone i componenti lungo i bordi e al centro
  - **`GridLayout`**, che dispone i componenti in una griglia m x n
  - **`GridBagLayout`**, che dispone i componenti in una griglia m x n flessibile
    - righe e colonne a dimensione variabile
    - molto flessibile e potente, ma difficile da usare
  - Altri layout manager (si veda la documentazione)

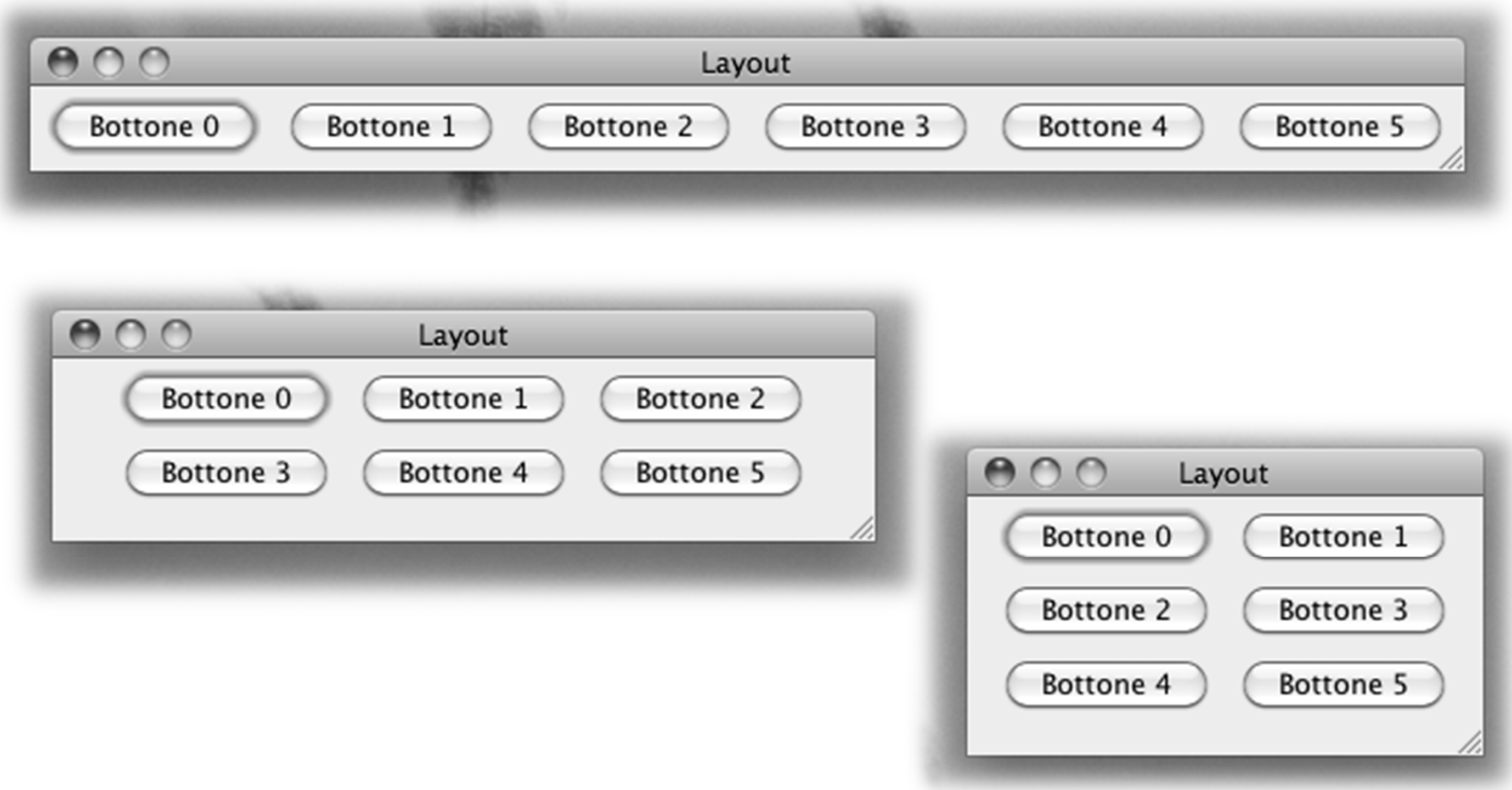
Posizione assoluta

- Per mettere i componenti in una posizione assoluta, è necessario togliere ogni layout manager

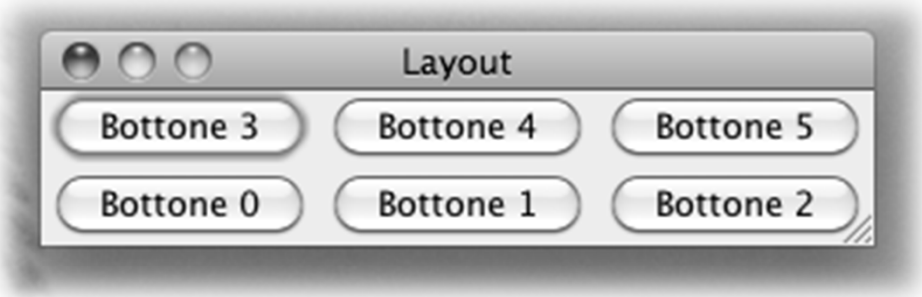
setLayout(null);

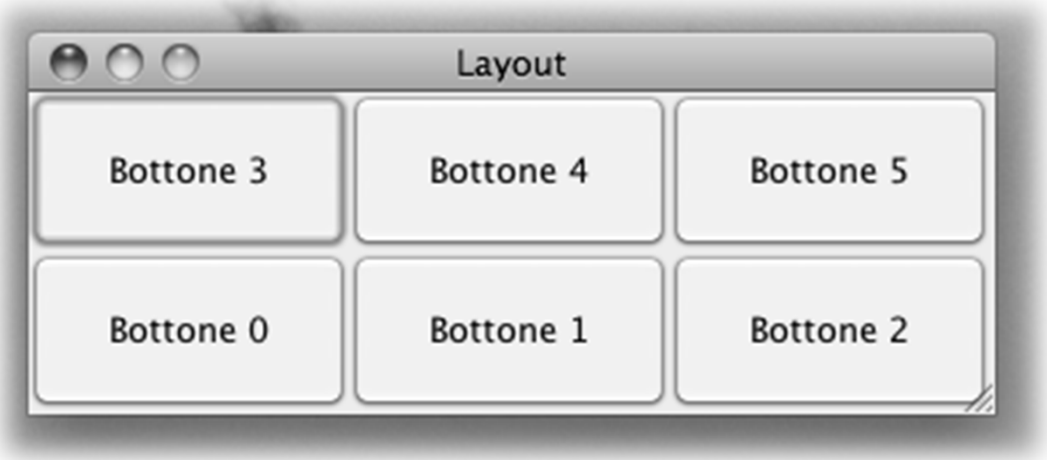
- Il rischio è quello avere risultati diversi su piattaforme (o risoluzioni) diverse

FlowLayout

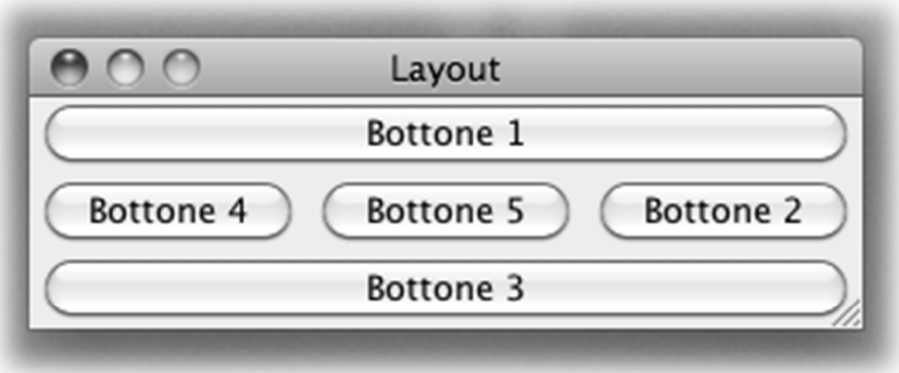

Layout
Bottone 0
Bottone 1
Bottone 2
Bottone 3
Bottone 4
Bottone 5
Layout
Bottone 0
Bottone 1
Bottone 2
Bottone 3
Bottone 4
Bottone 5
Layout
Bottone 0
Bottone 1
Bottone 2
Bottone 3
Bottone 4
Bottone 5

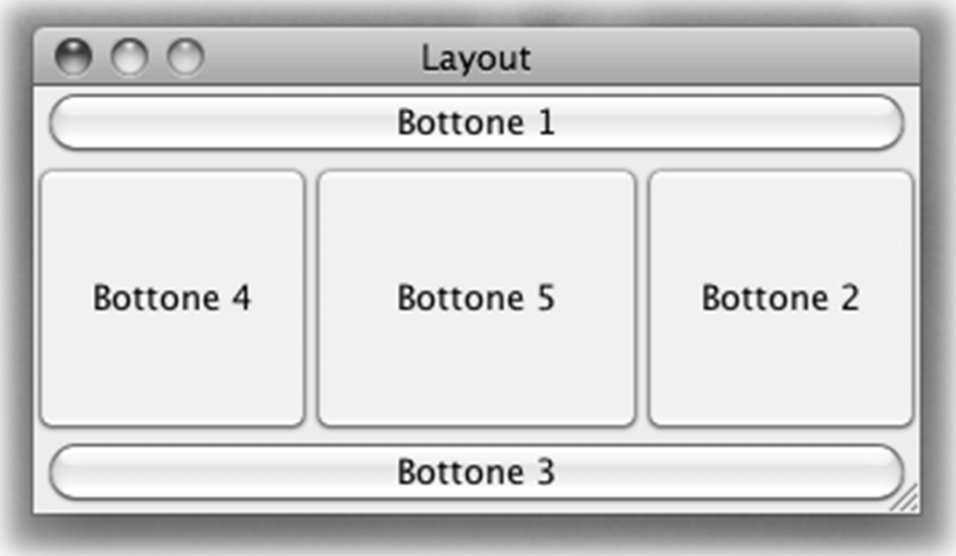
GridLayout 2 x 3





BorderLayout





BorderLayout con pannelli